An Algorithm for Computing Semi-Stable Semantics *

Martin Caminada

Utrecht University / University of Luxembourg

Abstract. The semi-stable semantics for formal argumentation has been introduced as a way of approximating stable semantics in situations where no stable extensions exist. Semi-stable semantics can be located between stable semantics and preferred semantics in the sense that every stable extension is a semi-stable extension and every semi-stable extension is a preferred extension. Moreover, in situations where at least one stable extension exists, the semi-stable extensions are equal to the stable extensions. In this paper we provide an outline of an algorithm for computing the semi-stable extensions, given an argumentation framework. We show that with a few modifications, the algorithm can also be used for computing stable and preferred semantics.

1 Introduction

Formal argumentation, as a technique for defeasible entailment, has gained popularity since it combines a relatively easy to understand and human-style approach to reasoning with the mathematical rigidness that is required for software implementation [1]. It is also an interesting observation that many formalisms for nonmonotonic reasoning can be expressed as instances of formal argumentation [2].

Formal argumentation, in its most abstract form, is done using a set of abstract arguments and a defeat relation between these arguments. Since an argument A may be defeated by another argument B which may in its turn be defeated by a third argument C, the status of A (whether it can be accepted or not) partly depends on the status of C. Thus, what is needed is an overall criterion for determining which of the arguments can be considered to be ultimately justified. Several of such criteria have been proposed, the most well-known of these are grounded, preferred and stable semantics [2]. A relatively new proposal is semi-stable semantics [3]. Semi-stable semantics can be placed between stable semantics and preferred semantics, as every stable extension is also a semi-stable extension and every semi-stable extension is also a preferred extension. Moreover, semi-stable semantics can be seen as a way of approximating stable semantics in situations where no stable extensions exist.

In this paper we present an algorithm for computing all semi-stable extensions of an argumentation framework. In order to keep the discussion brief, full formal proofs are provided in a seperate technical report [4].

* This work was sponsored by the Netherlands Organization for Scientific Research (NWO). We also thank Newres al Haider for helping to develop the initial idea.

2 Argument-Based Semantics

In this section, we provide a brief introduction on argument based semantics and the position of semi-stable semantics.

Definition 1. *An* argumentation framework *is a pair* (Ar, def) *where* Ar *is a finite set of arguments and* $def \subseteq Ar \times Ar$.

We say that an argument A *defeats* an argument B iff $(A, B) \in def$.
An argumentation framework can be represented as a directed graph in which the arguments are represented as nodes and the defeat relation is represented as arrows. In several examples throughout this paper, we will use this graph representation.
The shorthand notation A^+ and A^- stands for, respectively, the set of arguments defeated by A and the set of arguments that defeat A. Likewise, if $\mathcal{A}rgs$ is a set of arguments, then we write $\mathcal{A}rgs^+$ for the set of arguments that is defeated by at least one argument in $\mathcal{A}rgs$, and $\mathcal{A}rgs^-$ for the set of arguments that defeat at least one argument in $\mathcal{A}rgs$. In the definition below, $F(\mathcal{A}rgs)$ stands for the set of arguments that are acceptable in the sense of [2].

Definition 2 (defense / conflict-free). *Let* $A \in Ar$ *and* $\mathcal{A}rgs \subseteq Ar$.
We define A^+ *as* $\{B \mid A \text{ def } B\}$ *and* $\mathcal{A}rgs^+$ *as* $\{B \mid A \text{ def } B \text{ with } A \in \mathcal{A}rgs\}$.
We define A^- *as* $\{B \mid B \text{ def } A\}$ *and* $\mathcal{A}rgs^-$ *as* $\{B \mid B \text{ def } A \text{ with } A \in \mathcal{A}rgs\}$.
$\mathcal{A}rgs$ *is* conflict-free *iff* $\mathcal{A}rgs \cap \mathcal{A}rgs^+ = \emptyset$. $\mathcal{A}rgs$ defends *an argument* A *iff* $A^- \subseteq \mathcal{A}rgs^+$. *We define the function* $F : 2^{Ar} \to 2^{Ar}$ *as* $F(\mathcal{A}rgs) = \{A \mid A \text{ is defended by } \mathcal{A}rgs\}$.

In the definition below, definitions of grounded, preferred and stable semantics are described in terms of complete semantics, which has the advantage of making the proofs in the remainder of this paper more straightforward. These descriptions are not literally the same as the ones provided by Dung [2], but as was first stated in [5], these are in fact equivalent to Dung's original versions of grounded, preferred and stable semantics.

Definition 3 (acceptability semantics). *A conflict-free set* $\mathcal{A}rgs$ *of arguments is called*
- *an* admissible *set iff* $\mathcal{A}rgs \subseteq F(\mathcal{A}rgs)$.
- *a* complete *extension iff* $\mathcal{A}rgs = F(\mathcal{A}rgs)$.
- *a* grounded *extension iff* $\mathcal{A}rgs$ *is the minimal complete extension.*
- *a* preferred *extension iff* $\mathcal{A}rgs$ *is a maximal complete extension.*
- *a* stable *extension iff* $\mathcal{A}rgs$ *is a complete extension that defeats every argument in* $Ar \backslash \mathcal{A}rgs$.
- *a* semi-stable *extension iff* $\mathcal{A}rgs$ *is a complete extension where* $\mathcal{A}rgs \cup \mathcal{A}rgs^+$ *is maximal (w.r.t. set-inclusion)*

In [3] it is proved that every stable extension is also a semi-stable extension, and the every semi-stable extension is also a preferred extension. Moreover, it is observed that if the argumentation framework has at least one stable extension, then the set of semi-stable extensions is equal to the set of stable extensions. That is, when at least one stable extension exists, then stable semantics and semi-stable semantics coincide.

3 A Brief Introduction to Argument Labellings

The concepts of admissibility, as well as those of complete, grounded, preferred, stable or semi-stable semantics were originally stated in terms of sets of arguments. It is equally well possible, however, to express these concepts using *argument labellings*. This approach was originally proposed by Pollock [6] and has recently been extended by Caminada [5]. The idea of a labelling is to associate with each argument exactly one label, which can either be `in`, `out` or `undec`. The label `in` indicates that the argument is explicitly accepted, the label `out` indicates that the argument is explicitly rejected, and the label `undec` indicates that the status of the argument is undecided, meaning that one abstains from an explicit judgement whether the argument is `in` or `out`.

Definition 4. *A* labelling *is a function* $\mathcal{L} : Ar \longrightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$.

We write $\texttt{in}(\mathcal{L})$ for $\{A \mid \mathcal{L}(A) = \texttt{in}\}$, $\texttt{out}(\mathcal{L})$ for $\{A \mid \mathcal{L}(A) = \texttt{out}\}$ and $\texttt{undec}(\mathcal{L})$ for $\{A \mid \mathcal{L}(A) = \texttt{undec}\}$. Sometimes, we write a labelling $\mathcal{L}$ as a triple $(\mathcal{A}rgs_1, \mathcal{A}rgs_2, \mathcal{A}rgs_3)$ where $\mathcal{A}rgs_1 = \texttt{in}(\mathcal{L})$, $\mathcal{A}rgs_2 = \texttt{out}(\mathcal{L})$ and $\mathcal{A}rgs_3 = \texttt{undec}(\mathcal{L})$. We distinguish three special kinds of labellings. The *all-in labelling* is a labelling that labels every argument `in`. The *all-out labelling* is a labelling that labels every argument `out`. The *all-undec labelling* is a labelling that labels every argument `undec`.

Definition 5. *Let $\mathcal{L}$ be a labelling and A be an argument. We say that:*

1. *A is* illegally `in` *iff A is labelled* `in` *but not all its defeaters are labelled* `out`
2. *A is* illegally `out` *iff A is labelled* `out` *but does not have a defeater labelled* `in`
3. *A is* illegally `undec` *iff A is labelled* `undec` *but either all its defeaters are labelled* `out` *or it has a defeater that is labelled* `in`.

We say that a labelling has no illegal arguments *iff there is no argument that is illegally* `in`*, illegally* `out` *or illegally* `undec`*. We say that an argument is* legally `in` *iff it is labelled* `in` *and is not illegally* `in`*. We say that an argument is* legally `out` *iff it is labelled* `out` *and is not illegally* `out`*. We say that an argument is* legally `undec` *iff it is labelled undec and is not illegally undec.*

Definition 6. *An* admissible labelling *is a labelling without arguments that are illegally* `in` *and without arguments that are illegally* `out`.

Definition 7. *A* complete labelling *is a labelling without arguments that are illegally* `in`*, without arguments that are illegally* `out` *and without arguments that are illegally* `undec`.

Definition 8. *Let $\mathcal{L}$ be a complete labelling. We say that $\mathcal{L}$ is a*

- grounded labelling *iff* $\texttt{in}(\mathcal{L})$ *is minimal (w.r.t. set inclusion).*
- preferred labelling *iff* $\texttt{in}(\mathcal{L})$ *is maximal (w.r.t. set inclusion).*
- stable labelling *iff* $\texttt{undec}(\mathcal{L}) = \emptyset$.
- semi-stable labelling *iff* $\texttt{undec}(\mathcal{L})$ *is minimal (w.r.t. set inclusion).*

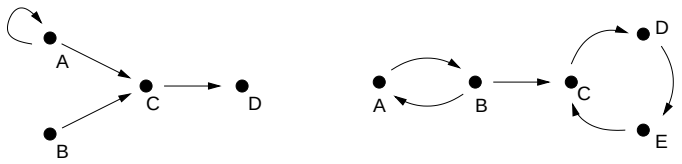

Fig. 1. Two argumentation frameworks.

As an illustration of how the various types of labellings can be applied, consider the two examples in Figure 1. For the example at the left hand side of Figure 1, there exists just one complete labelling: $(\{B, D\}, \{C\}, \{A\})$, which is then automatically also grounded, preferred and semi-stable. The example at the left hand side does not have any stable labellings. For the example at the right hand side of Figure 1, there exist three complete labellings: $(\emptyset, \emptyset, \{A, B, C, D, E\})$, $(\{A\}, \{B\}, \{C, D, E\})$ and $(\{B, D\}, \{A, C, E\}, \emptyset)$. The first labelling is the grounded labelling. The second and third labellings are both preferred labellings. The third labelling is also a stable and semi-stable labelling.
As for the admissible labellings, it should be mentioned that each complete labelling is also an admissible labelling. However, sometimes there exist admissible labellings that are not complete. Two examples of such labellings for the example at the left hand side of Figure 1 are $(\{B\}, \emptyset, \{A, C, D\})$ and $(\{B\}, \{C\}, \{A, D\})$.
It is interesting to notice that an admissible labelling actually corresponds with the notion of an admissible set.

Theorem 1. *Let (Ar, def) be an argumentation framework and $\mathcal{A}rgs \subseteq Ar$. $\mathcal{A}rgs$ is an admissible set iff there exists an admissible labelling $\mathcal{L}$ with $\mathtt{in}(\mathcal{L}) = \mathcal{A}rgs$.*

The validity of Theorem 1 can be seen as follows. If $\mathcal{A}rgs$ is an admissible set, then a labelling $\mathcal{L}$ with $\mathtt{in}(\mathcal{L}) = \mathcal{A}rgs$, $\mathtt{out}(\mathcal{L}) = \mathcal{A}rgs^+$ and $\mathtt{undec}(\mathcal{L}) = Ar - \mathtt{in}(\mathcal{L}) - \mathtt{out}(\mathcal{L})$ is an admissible labelling. Similarly, if $\mathcal{L}$ is an admissible labelling then $\mathtt{in}(\mathcal{L})$ is conflict-free (otherwise at least one of the arguments in $\mathtt{in}(\mathcal{L})$ would be illegally `in`). It can then be verified that $\mathtt{in}(\mathcal{L})$ defends itself, due to the fact that $\mathcal{L}$ does not contain arguments that are illegally `in` or illegally `out`. We refer to [4] for a full proof.
The notion of a complete labelling then corresponds to Dung's notion of a complete extension.

Theorem 2. *Let (Ar, def) be an argumentation framework and $\mathcal{A}rgs \subseteq Ar$. $\mathcal{A}rgs$ is a complete extension iff there exists a complete labelling $\mathcal{L}$ with $\mathtt{in}(\mathcal{L}) = \mathcal{A}rgs$.*

The validity of Theorem 2 can be seen as follows. If $\mathcal{A}rgs$ is a complete extension then a labelling with $\mathtt{in}(\mathcal{L}) = \mathcal{A}rgs$, $\mathtt{out}(\mathcal{L}) = \mathcal{A}rgs^+$ and $\mathtt{undec}(\mathcal{L}) = Ar - \mathtt{in}(\mathcal{L}) - \mathtt{out}(\mathcal{L})$ is a complete labelling. Similarly, if $\mathcal{L}$ is a complete labelling then $\mathtt{in}(\mathcal{L})$ is at least an admissible set (this

follows from Theorem 1). It can then be verified that $\texttt{in}(\mathcal{L})$ defends exactly itself, due to the fact that $\mathcal{L}$ does not contain any arguments that are illegally `in`, illegally `out` or illegaly `undec`. Hence, $\texttt{in}(\mathcal{L})$ is a complete extension. Again, we refer to [4] for a full proof.
The notions of a grounded, preferred, stable and semi-stable labelling correspond to he notions of a grounded, preferred, stable and semi-stable extension, respectively.

Theorem 3. *A set $\mathcal{A}rgs$ of arguments is (1) a grounded extension iff there exists a grounded labelling $\mathcal{L}$ with $\texttt{in}(\mathcal{L}) = \mathcal{A}rgs$, (2) a preferred extension iff there exists a preferred labelling $\mathcal{L}$ with $\texttt{in}(\mathcal{L}) = \mathcal{A}rgs$, (3) a stable extension iff there exists a stable labelling $\mathcal{L}$ with $\texttt{in}(\mathcal{L}) = \mathcal{A}rgs$, and (4) a semi-stable extension iff there exists a semi-stable labelling $\mathcal{L}$ with $\texttt{in}(\mathcal{L}) = \mathcal{A}rgs$*

Before continuing with the backgrounds of the proposed algorithm, we first state a few useful properties of complete and admissible labellings.

Lemma 1. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two complete labellings of (Ar, def). It holds that $\texttt{in}(\mathcal{L}_1) \subseteq \texttt{in}(\mathcal{L}_2)$ iff $\texttt{out}(\mathcal{L}_1) \subseteq \texttt{out}(\mathcal{L}_2)$.*

Lemma 2. *Let $\mathcal{L}_1$ be an admissible labelling. There exists a preferred labelling $\mathcal{L}_2$ with $\texttt{in}(\mathcal{L}_1) \subseteq \texttt{in}(\mathcal{L}_2)$ and $\texttt{out}(\mathcal{L}_1) \subseteq \texttt{out}(\mathcal{L}_2)$.*

Lemma 3. *Let $\mathcal{L}$ be a preferred labelling and $\mathcal{L}'$ be an admissible labelling. It holds that:*
1. *if $\texttt{in}(\mathcal{L}) \subseteq \texttt{in}(\mathcal{L}')$ then $\texttt{in}(\mathcal{L}) = \texttt{in}(\mathcal{L}')$*
2. *if $\texttt{out}(\mathcal{L}) \subseteq \texttt{out}(\mathcal{L}')$ then $\texttt{out}(\mathcal{L}) = \texttt{out}(\mathcal{L}')$*

4 Formal Background of the Algorithm

Now that the preliminary concepts have been explained, it is time to treat the main question of how to compute, given an argumentation framework, all the semi-stable labellings. The idea is to do this by generating a set $\mathcal{L}abellings$ of admissible labellings that includes at least all preferred labellings. Since every semi-stable labelling is also a preferred labelling [3, 5], this means that $\mathcal{L}abellings$ also contains all semi-stable labellings. We then have to select those labellings in $\mathcal{L}abellings$ with minimal `undec` to obtain the final answer.
How does one generate an admissible labelling? A possible approach is to start with the all-in labelling (the labelling in which every argument is labelled `in`). This labelling trivially satisfies the absence of arguments that are illegally `out`. However, for an admissible labelling also the absence of arguments that are illegally `in` is required, and the all-in labelling may contain many arguments that are illegally `in`. This means we need a way of changing the label of an argument that is illegally `in`, preferrably without creating any arguments that are illegally `out`. This is done using a sequence of *transition steps*. A transition step basically takes an argument that is illegally `in` and relabels it to `out`. It then checks if, as a result of this, one or more arguments have become illegally `out`. If this is the case, then these arguments are relabelled to `undec`. More precisely, a transition step can be described as follows.

Definition 9. *Let $\mathcal{L}$ be a labelling and A an argument that is illegally* `in` *in $\mathcal{L}$. A* transition step *on A in $\mathcal{L}$ consists of the following:*

1. *the label of A is changed from* `in` *to* `out`
2. *for every $B \in \{A\} \cup A^+$, if B is illegally* `out`*, then change the label of B from* `out` *to* `undec`*.*

Theorem 4. *Each transition step preserves the absence of arguments that are illegally* `out`*.*

The validity of Theorem 4 follows directly from point 2 of Definition 9. A transition sequence starts with an initial labelling $\mathcal{L}_0$, on which a sequence of successive transition steps is applied.

Definition 10. *A* transition sequence *is a list $[\mathcal{L}_0, A_1, \mathcal{L}_1, A_2, \mathcal{L}_2, \ldots, A_n, \mathcal{L}_n]$ ($n \geq 0$) where each A_i ($1 \leq i \leq n$) is an argument that is illegally* `in` *in labelling $\mathcal{L}_{i-1}$ and every $\mathcal{L}_i$ is the result of doing a transition step of A_i on $\mathcal{L}_{i-1}$. A transition sequence is called* terminated *iff $\mathcal{L}_n$ does not contain any argument that is illegally* `in`*.*

As an illustration of how a transition sequence can be constructed, consider the example at the left hand side of Figure 2. Assume the initial situation is the all-in labeling $\mathcal{L}_0 = (\{A, B, C\}, \emptyset, \emptyset)$. In this labelling both B and C are illegally `in` since each of them has a defeater that is `in`, so they are both candidates for a transition step. If we select B for a transition step, then the result is a labelling $\mathcal{L}_1 = (\{A, C\}, \{B\}, \emptyset)$. This labelling does not contain any arguments that are illegally `in`, so the transition sequence $[\mathcal{L}_0, B, \mathcal{L}_1]$ is terminated. If, at the other hand, we select C for a transition step then the result is a labelling $\mathcal{L}'_1 = (\{A, B\}, \{C\}, \emptyset)$. This labelling still has an argument that is illegally `in` (B), so we perform another transition step that relabels B from `in` to `out`. However, as a result of doing that, C becomes illegally `out` since it has no longer a defeater that is `in`, so C is relabelled from `out` to `undec`. The transition step as a whole then yields $\mathcal{L}'_2 = (\{A\}, \{B\}, \{C\})$. This means that there exists a second terminated transition sequence $[\mathcal{L}_0, C, \mathcal{L}'_1, B, \mathcal{L}'_2]$.
Now consider the example at the right hand side of Figure 2. Again, assume that the initial labelling is the all-in labelling, so $\mathcal{L}_0 = (\{A, B, C\}, \emptyset, \emptyset)$. Here, all three arguments are `in`, so each of them can be selected for a transition step. Assume, without loss of generality, that A is selected for a transition step. This then yields a labelling $\mathcal{L}_1 = (\{B, C\}, \{A\}, \emptyset)$. In this labelling, only C is illegally `in` and can be selected for a transition step. During this transition step, after C is relabelled from `in` to `out`, A becomes illegally `out` and is therefore relabelled to `undec`. Thus, the transition step as a whole yields $\mathcal{L}_2 = (\{B\}, \{C\}, \{A\})$. In this labelling B is illegally `in` since it has a defeater (A) that is `undec`. Therefore, a transition step on B is performed during which B is relabelled from `in` to `out`. Directly after doing that, however, not only C is illegally `out` but also B itself is illegally `out`, so both of them are relabelled from `out` to `undec`. Thus, the transition step as a whole yields $\mathcal{L}_3 = \{\emptyset, \emptyset, \{A, B, C\})$. This means that there exists a terminated transition sequence $[\mathcal{L}_0, A, \mathcal{L}_1, C, \mathcal{L}_2, B, \mathcal{L}_3]$. It can be verified that in the example

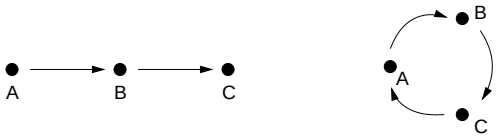

Fig. 2. Two argumentation frameworks.

at the right hand side of Figure 2 every terminated transition sequence that starts with the all-in labelling finishes with $\mathcal{L}_3$.
Since for any finite argumentation framework, only a finite number of successive transition steps can be performed, this means that (again for finite argumentation frameworks) each terminated transition sequence is finite. Furthermore, for any terminated transition sequence, the final labelling is an admissible labelling. This is because each transition step preserves the absence of arguments that are illegally `out` (Theorem 4) and after termination, we also do not have any arguments that are illegally `in`.

Theorem 5. *Let* $[\mathcal{L}_0, A_1, \mathcal{L}_1, A_2, \mathcal{L}_2, \ldots, A_n, \mathcal{L}_n]$ *(*$n \geq 0$*) be a terminated transition sequence where* $\mathcal{L}_0$ *is the all-in labelling. It holds that* $\mathcal{L}_n$ *is an admissible labelling.*

An interesting observation is that during the course of a transition sequence, the set of `in`-labelled arguments monotonically decreases and the set of `undec`-labelled arguments monotonically increases.

Proposition 1. *Let* $[\mathcal{L}_0, A_1, \mathcal{L}_1, \ldots, A_n, \mathcal{L}_n]$ *be a transition sequence. For any* $1 \leq i \leq n$ *it holds that* $\texttt{in}(\mathcal{L}_i) \subseteq \texttt{in}(\mathcal{L}_{i-1})$ *and* $\texttt{undec}(\mathcal{L}_i) \supseteq \texttt{undec}(\mathcal{L}_{i-1})$.

Proposition 1 is relevant with respect to the algorithm for generating the semi-stable labellings. Suppose that a previously generated terminated transition sequence yielded an admissible labelling $\mathcal{L}$ and we are currently expanding a transition sequence $[\mathcal{L}_0, A_1, \mathcal{L}_1, \ldots, A_i, \mathcal{L}_i]$. If $\texttt{undec}(\mathcal{L}_i) \supsetneq \texttt{undec}(\mathcal{L})$ then we already know that the current transition sequence cannot yield a semi-stable labelling, since expanding it to a terminated transition sequence $[\mathcal{L}_0, A_1, \mathcal{L}_1, \ldots, A_n, \mathcal{L}_n]$ results in a labelling $\mathcal{L}_n$ with $\texttt{undec}(\mathcal{L}_n) \supsetneq \texttt{undec}(\mathcal{L})$ (this follows from Proposition 1 and the fact that $\texttt{undec}(\mathcal{L}_i) \supsetneq \texttt{undec}(\mathcal{L})$). We then might as well stop expanding the current transition sequence and instead try another possibility.
We define *Labellings* as the set of all final labellings from terminated transition sequences that start from the all-in labelling.
As we have now obtained that the result of any terminated transition sequence starting from the all-in labelling is an admissible labelling, it directly follows that each element of *Labellings* is an admissible labelling. The next step, then, is to examine whether each semi-stable labelling will be an element of *Labellings*. If this is the case, then we can simply determine the semi-stable labellings as those elements of *Labellings* where

undec is minimal. It turns out that this is indeed the case. This can roughly be seen as follows. Let $\mathcal{L}$ be a preferred labelling. We now construct a transition sequence that yields $\mathcal{L}$. This is done in two phases. The first phase is to perform a sequence of transition steps, starting from the all-in labelling, on each argument that is labelled out in $\mathcal{L}$. This yields a labelling $\mathcal{L}'$ with $\texttt{out}(\mathcal{L}') = \texttt{out}(\mathcal{L})$, $\texttt{undec}(\mathcal{L}') = \emptyset$ and $\texttt{in}(\mathcal{L}') \supseteq \texttt{in}(\mathcal{L})$. Then, during the second phase, we continue to perform transition steps, starting from $\mathcal{L}'$, until we have reached termination; that is, until there are no arguments that are illegally in anymore, yielding a labelling $\mathcal{L}''$. It can be verifed that this does not change he arguments that are out in $\mathcal{L}'$. Also, it cannot change the arguments that are in in $\mathcal{L}$, since these are legally in in $\mathcal{L}'$. That is, we have that $\texttt{out}(\mathcal{L}'') = \texttt{out}(\mathcal{L})$ and $\texttt{in}(\mathcal{L}'') \supseteq \texttt{in}(\mathcal{L})$. From the fact that $\mathcal{L}$ is a preferred labelling, it follows that (Lemma 3) $\texttt{in}(\mathcal{L}'')$ cannot be a strict superset of $\texttt{in}(\mathcal{L})$. Therefore, we have that $\texttt{in}(\mathcal{L}'') = \texttt{in}(\mathcal{L})$. From the fact that $\texttt{in}(\mathcal{L})$ and $\texttt{out}(\mathcal{L}'') = \texttt{out}(\mathcal{L})$ it follows that $\texttt{undec}(\mathcal{L}'') = \texttt{undec}(\mathcal{L})$, which implies that $\mathcal{L}'' = \mathcal{L}$. This leads to the following theorem.

Theorem 6. *Let $\mathcal{L}$ be a preferred labelling. There exists a transition sequence $[\mathcal{L}_0, A_1, \mathcal{L}_1, \ldots, A_n, \mathcal{L}_n]$ with $\mathcal{L}_0$ the all-in labelling and $\mathcal{L}_n = \mathcal{L}$.*

From the fact that each semi-stable labelling is also a preferred labelling, it then follows that for each semi-stable labelling, there exists a transition sequence that yields it.

5 Optimizing the Algorithm

As was shown in Section 4, for the example at the left hand side of Figure 2 there are two terminated transition sequences starting from the all-in labelling: one that yields $(\{A, C\}, \{B\}, \emptyset)$ and one that yields $(\{A\}, \{B\}, \{C\})$. This is because starting from the all-in labelling, we have two choices of arguments to do a transition step on: B or C, since both of them are illegally in in $\mathcal{L}_0$ (the all-in labelling). If we choose B we will finally end up with a complete labelling, but if we choose C then we will ultimately end up with a labelling that is admissible but not complete (and therefore also not preferred or semi-stable). An interesting question, therefore, is whether there is a way of avoiding such non-complete results by carefully choosing the right arguments to do the transition steps on. While in general this question is difficult to answer, we do propose a simple guideline that is helpful in many cases: choose an argument that is *superillegally* in to do a transition step on, if such an argument is available.

Definition 11. *Let $\mathcal{L}$ be a labelling of (Ar, def). An argument A is* superillegally in *in $\mathcal{L}$ iff A is labelled* in *by $\mathcal{L}$ and is defeated by an argument that is legally* in *in $\mathcal{L}$ or* undec *in $\mathcal{L}$.*

It directly follows that if an argument is superillegally in in $\mathcal{L}$, then it is also illegally in in $\mathcal{L}$. The converse, however, may not be the case. As an example, consider again the example at the left hand side of Figure

2. With the all-in labelling, A is legally `in`, B and C are illegally `in`, and only B is superillegally `in`. Thus, it makes sense to select B to do a transition step on.
The reason why arguments that are superillegally `in` are such good candidates to perform a transition step on is that an argument that is superillegally `in` will stay illegally `in` (although it may not necessarily stay *super*illegally `in`) throughout the transition sequence, until a transition step is done on it. Thus, we might as well perform a transition step on the superillegal argument as soon as possible, since this prevents us from doing things we later regret (like performing a transition step on argument C).

Theorem 7. *Let $\mathcal{L}_0$ be a labelling where argument A is superillegally* `in` *and $[\mathcal{L}_0, A_1, \mathcal{L}_1, \ldots, A_n, \mathcal{L}_n]$ be a transaction sequence where no transaction step is performed on A (that is: $A \notin \{A_1, \ldots, A_n\}$). It holds that A is illegally* `in` *in $\mathcal{L}_n$.*

From Theorem 7 it follows that it may be a good strategy to select an argument that is superillegally `in` to do a transition step on, whenever such an argument is available. An interesting question is how such a strategy would affect the results that were obtained earlier regarding correctness (each transition sequence terminates with an admissible labelling) and completeness (for each preferred labelling, there exists a transition sequence that produces this labelling).
As for correctness, the situation does not change. The result of a terminated transition sequence is always an admissible labelling, regardless of which strategy was used to select the arguments to do transition steps on. In [4] it is explained that the new strategy also does not affect the completeness of the algorithm. That is, if we consequently choose an (arbitrary) superillegal argument to do a transition step on whenever such an argument is available, then we are still able to produce all preferred labellings, and therefore also all semi-stable labellings.

6 The Actual Algorithm

Since the algoritm starts with the labelling in which every argument is labelled `in`, we assume the presence of the constant `all_in`, which stands for the all-in labelling. There is one global variable (`pot_semi-stables`) which stands the potential semi-stable labellings, that is, the admissible labellings with minimal `undec` that have been found until now. If, during the search algorithm, one finds that the current labelling is worse (that is: it has a proper superset of `undec` labelled arguments) than an admissible labelling found earlier, then it is time to stop evaluating the current transition sequence, since its final result will not be semi-stable anyway. If there is no argument that is illegally `in` then we are at the end of a terminated transition sequence and have obtained an admissible labelling. From the previous check, we already know that this admissible labelling is not any worse than what we already have found (it does not have a proper superset of `undec` labelled arguments compared to a previously

computed admissible labelling), so we add it to the set of potential semi-stable labellings (`pot_semi-stables`). We then have to check if we found something that is actually *better* than what we found earlier. If so, we need to delete some of the old results (remove it from `pot_semi-stables`). If we have not reached the end of a terminated transition sequence, then there is at least one argument that is still illegally `in`. We then distinguish two cases. If there is at least one argument that is superillegally `in` then go for the argument that is superillegally `in`. There is no need to be selective; any argument that is superillegally `in` will do for a transition step. If, however, there is no argument that is superillegally `in` then we have to try each argument that is "normally" illegally `in`.

```
01. pot_semi-stables = ∅; find_semi-stables(all-in);
02. print pot_semi-stables; end;
03.
04. procedure find_semi-stables(ℒ)
05.    # if we have something worse than found earlier,
06.    # then prune the search tree and backtrack
07.    if ∃ℒ′ ∈ pot_semi-stables: undec(ℒ′) ⊊ undec(ℒ) then return;
08.    # now see if the transition sequence has terminated
09.    if ℒ does not have an argument that is illegally in then
10.      for each ℒ′ ∈ pot_semi-stables
11.        # if old result is worse than new labelling: remove
12.        if undec(ℒ) ⊊ undec(ℒ′) then
13.          pot_semi-stables := pot_semi-stables - ℒ′;
14.        endif;
15.      endfor;
16.      # add our newly found labelling as a candidate; we already
17.      # know that it is not worse than what we already have
18.      pot_semi-stables := pot_semi-stables ∪ ℒ;
19.      return; # we are done with this one; try next possibility
20.    else
21.      if ℒ has an argument that is superillegally in then
22.        A := some argument that is superillegally in in ℒ;
23.        find_semi-stables(transition_step(A, ℒ));
24.      else
25.        for each argument A that is illegally in in ℒ
26.          find_semi-stables(transition_step(A, ℒ));
27.        endfor;
28.      endif;
29.    endif;
30. endproc;
```

7 Discussion

It is interesting to observe that the algorithm stated in Section 6 can also be used to calculate, respectively, stable semantics and preferred semantics, by applying a few changes.
For stable semantics, the modification is quite straightforward. Basically, the idea (Definition 8) is only to yield labellings without `undec`-labelled

arguments. For this, we have to stop expanding a transition sequence as soon as an `undec`-labelled argument is produced. Therefore, we have to replace line 7 by `if` $\texttt{undec}(\mathcal{L}) \neq \emptyset$ `then return`.
Furthermore, we do not have to compare the sets of `undec`-labelled arguments of the previous results with the current result, so the lines 10 until 15 can be removed. Then, after renaming the variable `pot_semi-stables` to `stables` and renaming the procedure `find_semi-stables` to `find_stables`, the modifications are finished and the result is an algorithm that calculates all stable extensions of an argumentation framework.
For preferred semantics, the modification is slightly different. The idea is that we still have to check for a condition that allows us to cut off the current transition sequence once we know that it will not yield a useful result. For semi-stable semantics, it can be observed that the set of `undec`-labelled arguments keeps *increasing* as the transition sequence progresses (Proposition 1). For preferred semantics, it can be observed that the set of `in`-labelled arguments keeps *decreasing* as the transition sequence progresses (Proposition 1). In both cases, there may come a point where the current transition sequence becomes worse than a result found earlier, which means we might as well stop expanding it and instead backtrack to another possibility.
The modification for preferred semantics is done as follows. First, the variable `pot_semi-stables` to renamed as `pot_preferreds` and the procedure `find_semi-stables` is renamed as `find_preferreds`. Line 7 is replaced by: `if` $\exists\mathcal{L}' \in$ `pot_preferreds:` $\texttt{in}(\mathcal{L}') \supsetneq \texttt{in}(\mathcal{L})$ `then return;`
Line 12 is replaced by: `if` $\texttt{in}(\mathcal{L}) \supsetneq \texttt{in}(\mathcal{L}')$ `then`
The result, then, is an algorithm that calculates, given an argumentation framework, all preferred extensions.
In [5], it was first examined how argument labellings are related to the traditional Dung-style argument semantics.It was found that a complete labelling has a maximal set of `in`-labelled arguments iff it has a maximal set of `out`-labelled arguments. In both cases, the labelling corresponds with a preferred extension. Furthermore, it was found that a complete labelling has a minimal set of `in`-labelled arguments iff it has a minimal set of `out`-labeled arguments iff it has a maximal set of `undec`-labelled arguments. In all three cases, the labelling corresponds with the grounded extension. The only option left to be examined consists of the labellings where the set of `undec`-labelled arguments is minimal. It turned out that these did not correspond with any well-known semantics, and this is how semi-stable semantics was discovered [3, 5]. Thus, one can perhaps think of semi-stable semantics as a missing link in the traditional hierarchy of argumentation semantics.
Nevertheless, semi-stable semantics is more than just a purely theoretical notion. Stable semantics, despite its property of the potential absence of stable extensions, is still being used for the purpose of constraint satisfaction in fields like answer set programming [7]. The idea is that a problem is specified in a declarative way and that the set of potential solutions then corresponds with the stable models of the thus described problem. In cases where no solutions exists, there should therefore also not exist any stable models. Thus, the absence of stable models (or extensions) is not always an undesirable property. This does assume, however, that

the original problem was encoded in a way that is perfectly correct. If, for instance, an answer set program contains an error, then the result may well be a total absence of stable models, which is a situation that can be notoriously hard to debug. With semi-stable semantics, however, one obtains one or more models that can serve as a starting point to examine where things went wrong. For instance, consider the example at the left hand side of Figure 1. It has no stable extensions and its (only) semi-stable extension is $(\{B, D\}, \{C\}, \{A\})$. The fact that A is labelled `undec` can be seen as an indication of what is "wrong" in this argumentation framework from the perspective of stable semantics. Similarly, if there exists an odd loop that causes the absence of stable models, then this odd loop is flagged `undec` by a semi-stable model. Thus, semi-stable semantics can give a good indication of where to start debugging if no stable model exists. Semi-stable semantics does a better job here than, for instance, preferred semantics. This is because there can be non-stable preferred models (like $(\{A\}, \{B\}, \{C, D, E\})$) even in cases where stable models (like $(\{B, D\}, \{A, C, E\}, \emptyset)$ do exist. With semi-stable semantics, one obtains non-stable semi-stable models only if there is a real problem that prevents the existence of stable models.

One particularly interesting application of semi-stable semantics would be answer set programming and other forms of logic programming that use the stable model semantics. At the time of writing, the author is exploring the possibilities of applying semi-stable semantics to logic programming and answer set programming. We believe this would be a useful approach for analyzing programs for which no stable models exist.

References

1. Vreeswijk, G.: An algorithm to compute minimally grounded and admissible defence sets in argument systems. In Dunne, P., Bench-Capon, T., eds.: Computational Models of Argument; Proceedings of COMMA 2006, IOS (2006) 109–120
2. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77** (1995) 321–357
3. Caminada, M.: Semi-stable semantics. In Dunne, P., Bench-Capon, T., eds.: Computational Models of Argument; Proceedings of COMMA 2006, IOS Press (2006) 121–130
4. Caminada, M.: An algorithm for computing semi-stable semantics. Technical Report Technical Report UU-CS-2007-010, Utrecht University (2007) http://www.cs.uu.nl/~martinc/algorithm_techreport.pdf.
5. Caminada, M.: On the issue of reinstatement in argumentation. In Fischer, M., van der Hoek, W., Konev, B., Lisitsa, A., eds.: Logics in Artificial Intelligence; 10th European Conference, JELIA 2006, Springer (2006) 111–123 LNAI 4160.
6. Pollock, J.L.: Cognitive Carpentry. A Blueprint for How to Build a Person. MIT Press, Cambridge, MA (1995)
7. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9**(3/4) (1991) 365–385